Software Requirements Specification

for

Payload Launch Control Simulator

Version 1.0

James Furtado

1. Introduction

1.1 Purpose

This document outlines the requirements for the Payload Launch Control Simulator (PLCS), a C++ simulation of a submarine payload launch system. It includes both functional and non-functional requirements and will guide the design, implementation, and testing of the project.

1.2 Intended Audience

This document is meant for:

- **Myself, the developer**
  - To practice C++, explore defense-related systems, and get hands-on with systems engineering ideas.
- **Defense contractor hiring managers or resume reviewers**
  - To show my understanding of defense workflows and interest in the field through this relevant and well-structured project.
- **General users**
  - Anyone who wants to try out the simulator or use parts of it as a reference.

1.3 Scope

This system simulates the software side of a submarine payload launch process. It includes concepts like arming the payload, running safety checks, environmental constraints, and giving launch commands through a UI. Development is done with a focus on simulating safety, correctness, and clear operator flow.

1.4 Definitions, Acronyms, and Abbreviations

- PLCS: Payload Launch Control Simulator
- UI: User Interface
- FR: Functional Requirement
- NFR: Non-Functional Requirement
- Operator: User giving commands (like arming or launching)
- Safety System: The subsystem that checks if safe launch conditions are met
- Simulation Engine: The backend that handles system logic and state changes

2. Overall Description

2.1 Product Perspective

This is a standalone C++ simulation of a submarine payload launch control system. It models how an operator would interact with a launch UI, handle safety checks, and issue launch commands. No hardware connections are required– the project is rather created to feel like a real mission control system.

2.2 User Roles and Characteristics

- **Operator**
  - Uses the UI to arm payloads, check safety status, and launch. They have to wait for valid launch conditions, which can change over time.
- **Safety System**
  - Checks for safety violations and blocks launch if anything is off.
- **Simulation Engine**
  - Runs the system logic and keeps track of the current state. The engine includes environmental constraints like depth, position, and system status that can prevent launches. It's the operator's job to find valid windows to proceed.

2.3 Assumptions and Dependencies

- Software-only simulation; no hardware involved
- Single operator in control
- Runs on desktop platforms (Windows/Linux/macOS)
- Built with modern C++ (C++17 or later)
- Basic command-center style GUI
- Uses CMake as the build system for cross-platform portability and ease of setup

3. Functional Requirements

FR-1: The system shall allow the operator to arm the payload through the UI.

FR-2: The system shall allow the operator to issue a launch command through the UI.

FR-3: The system shall block launch commands if any safety conditions are not met.

FR-4: The system shall display the current status of all safety conditions in real-time.

FR-5: The system shall simulate environmental constraints such as submarine depth, position, and system status.

FR-6: The system shall update and display environmental values that affect whether a launch is permitted.

FR-7: The system shall log key actions and events such as arming, launch attempts, blocked launches, and state changes.

FR-8: The system shall reset to a safe state after each launch or failed launch attempt.

FR-9: The system shall allow the simulation state to be reset manually by the operator.

4. Non-Functional Requirements

NFR-1: The system shall respond to operator commands within 200 milliseconds.

NFR-2: The system shall display all status updates and safety checks in real-time.

NFR-3: The system UI shall be simple, clear, and focused on operator flow.

NFR-4: The system shall include automated tests for core functionality.